CSC 282 - Fall 2015
http://www.cs.rochester.edu/~stefanko/Teaching/15CS282/

Name:

Honor Pledge (following http://www.rochester.edu/college/honesty/policy.html#pledge):

I affirm that I will not give or receive any unauthorized help on this exam, and that all work will be my own.

____
(signature/date)

problem 1
problem 2
problem 3
TOTAL (non-bonus)
problem 4 (bonus)

do not write on this page

1. (40 POINTS) We have n items. The weight of the i-th item is $W[i]$, the value of the i-th item is $V[i]$, and the volume of the i-th item is $B[i]$. Assume that $W[i]$'s and $B[i]$'s are integers and $V[i]$'s are real numbers. We have two robbers:

   - "Weak" - with weight limit C and unlimited volume capacity,
   - "Small" - with volume limit D and unlimited weight capacity.

   That is, "Weak" can carry a subset of the items with total weight at most C and "Small" can carry a subset of the items with total volume at most D. There is no limit on the total weight of items "Small" can carry and there is no limit on the total volume of items "Weak" can carry.

   The robbers colluded and want to steal a subset of the items with the maximum possible total value[1].

   We will solve the problem using dynamic programming. Let $K[x, y, k]$ be the solution of the sub-problem restricted to the first k items where "Weak" has weight limit x and "Small" has volume limit y. Give a formula (or a piece of pseudo-code) for $K[x, y, k]$ in terms of $K[\cdot, \cdot, k-1]$ (where $\cdot$'s are to be replaced by appropriate expressions). Clearly describe what kinds of optimal solutions do different parts of your expression address. (Do NOT worry about printing the final solution.)

[1] That is, we are maximizing $\sum_{i \in S} V[i] + \sum_{i \in T} V[i]$ over $S, T \subseteq \{1, \ldots, n\}$ such that $S \cap T = \emptyset$ and $\sum_{i \in S} W[i] \leq C$ and $\sum_{i \in T} B[i] \leq D$. Here S is the set of items given to "Weak" and T is the set of items given to "Small".

2. (40 POINTS) Let $A = a_1, a_2, \ldots, a_n$ be a sequence of real numbers (any real numbers, including negative numbers are allowed). A selection of the numbers is called **valid** if

   - no three consecutive numbers are selected, AND
   - from each three consecutive numbers at least one is selected.

   We want to find a valid selection of $a_1, \ldots, a_n$ with the maximum sum. We will solve the problem using dynamic programming. (We will only find the value of the optimal solution—do not worry about printing it.) Let

   - $T[k, 0, 0]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} and a_k are NOT chosen,
   - $T[k, 1, 1]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} and a_k are chosen,
   - $T[k, 0, 1]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} is NOT chosen and a_k is chosen,
   - $T[k, 1, 0]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} is chosen and a_k are NOT chosen.

   Give expressions for $T[k, 0, 0]$, $T[k, 1, 1]$, $T[k, 0, 1]$, $T[k, 1, 0]$ in terms of $T[k-1, \cdot, \cdot]$ and a_k (where $\cdot$'s are to be replaced by appropriate expressions). Do NOT use any other entries in T, besides $T[k-1, \cdot, \cdot]$ in your expression (in particular, do not use $T[k-2, \cdot, \cdot]$ in your expression for $T[k, \cdot, \cdot]$).

   $T[k, 0, 0] =$

   $T[k, 1, 1] =$

   $T[k, 0, 1] =$

   $T[k, 1, 0] =$

3. (40 POINTS) Let $A = a_1, a_2, \ldots, a_n$ be a sequence of real numbers. We will call a subsequence[2] of A **good** if from each three consecutive numbers at least one is selected[3]. We want to find the length of the longest increasing good subsequence of A.

   Example 1: Let $A = 30, 31, 4, 15, 6, 1, 2, 7, 3, 4, 8, 9$. The answer is 5—we can take the following valid increasing subsequence of length 5: $30, 31, \underline{4}, 15, \underline{6}, 1, 2, \underline{7}, 3, 4, \underline{8}, \underline{9}$ (the numbers in the subsequence are underlined).

   Example 2: Let $A = 30, 31, 32, 7, 8, 1, 2, 3$. The answer is $-\infty$—there is no good increasing subsequence of A (since we have to take at least one of 30,30,31 and at least one of 1,2,3). (Since the maximum[4] of an empty set is $-\infty$ this is a good value to represent non-existence of a good increasing subsequence.)

   We will solve the problem using dynamic programming. Let $T[k]$ be the length of the longest increasing good subsequence of $a_1, \ldots, a_k$ that ends with a_k (if no such subsequence exists we take $T[k] = -\infty$). Give an expression (or a piece of code) that efficiently computes $T[k]$ from $T[1], \ldots, T[k-1]$ and $a_1, \ldots, a_k$.

[2] A **subsequence** of a sequence $a_1, \ldots, a_n$ is a sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_\ell}$ where $\ell \in \{0, \ldots, n\}$ and $1 \le i_1 < i_2 < \cdots < i_\ell \le n$.

[3] That is, for every $j \in \{1, \ldots, n-2\}$ we have $\{i_1, i_2, \ldots, i_\ell\} \cap \{j, j+1, j+2\} \neq \emptyset$.

[4] The maximum (for infinite sets supremum) of a set S of real numbers is the smallest $y^* \in \mathbb{R} \cup \{-\infty, \infty\}$ such that $(\forall x \in S)(x \le y^*)$. Smallest means: $(\forall y \in \mathbb{R} \cup \{-\infty, \infty\})(((\forall x \in S) x \le y) \implies y^* \le y)$.

4. (40 BONUS POINTS[5]) We have n coins with integer values $a_1, \ldots, a_n$ (each a_i corresponds to a physical coin). We have two children: Alice and Bob. We want to give some coins to Alice and some coins to Bob. The total value of coins we give to Alice must be equal to the total value of coins we give to Bob. We want to give them as many coins as possible (in total).

   Example: Suppose we have coins with values: 1,3,9,27,40,100,141. The optimal solution is to give 1,3,9,27 to one child and 40 to the other child (giving them 5 coins in total). (We could also give 1,40,100 to one child and 141 to the other one but this way we give them only 4 coins in total and hence this solution is NOT optimal.)

   Give a dynamic programming solution to this problem. Clearly define table (subproblems), recurrence, value of the optimal solution, and give a procedure to print the optimal solution.

[5] It is not recommended to work on this problem unless you solved the first 3 problems. The solutions of this problem will be held to a very high standard: quantities must be clearly defined, dynamic programming recurrences must be clearly argued to be correct, etc.